SCHEMA INTEGRATION USING METADATA

Michael Siegel
Stuart E. Madnick

October 1989 WP# 3092-89-MS

Schema Integration Using Metadata

Michael Siegel
Stuart E. Madnick
Sloan School of Management, E53-323
Massachusetts Institute of Technology
Cambridge, MA 02139
msiegel@sloan.mit.edu

Abstract

This report describes the use of metadata in the development and maintenance of a global schema interface to heterogeneous databases. First, we define metadata in terms of abstract data types and operators. Then, we show how metadata can be used in the interface between a database and an application. We generalize this use of metadata to simplify schema integration and semantic reconciliation between a global schema and an application. Metadata is also used to determine if changes in data semantics affect the data requirements of an application. In a heterogeneous system, metadata can be used to determine if an application is affected by changes in the semantics of the underlying databases. Finally, we show that, in some instances, when the global schema and the application are not semantically consistent, metadata can be used to modify the data so that it will be meaningful to the application.

Acknowledgments

The authors would like to thank the members of the Composite Information Systems Laboratory at the Sloan School of Management for the numerous discussions pertaining to this topic. In addition, discussions with the following persons were helpful to us as we identified and analyzed this problem; Sandy Heiler, John Mitchell, Robert Muller, Richard Shapiro and Richard Wang. This work was supported in part by Reuters and the International Financial Research Services Research Center at the Massachusetts Institute of Technology.

1 Introduction

The Composite Information Systems Laboratory (CISL) at the MIT Sloan School of Management has designed and developed a prototype system for access to multiple heterogeneous database systems, called the Composite Information System/Tool Kit (CIS/TK) [11]. CIS/TK, like many other heterogeneous database systems [1,3,10], presents the user with a global schema representing the integration of a set of local databases. Designers of these systems have examined schema integration in the presence local databases with static structure and semantics. However, to allow for greater local database autonomy, schema integration must be considered a dynamic problem. The global schema must be able to evolve to reflect changes in the structure and meaning of the underlying databases. If an application is affected by these changes, it must be alerted.

In this report we describe the use of metadata in the development and maintenance of the global schema. We describe how metadata can be used to simplify schema integration and semantic reconciliation between the global schema and an application. Semantic reconciliation is an important step in determining logical connectivity in a heterogeneous system [9]. Metadata can also be used to determine if an application running off the global schema is affected by changes in the semantics of the underlying databases. If the database and the application are not semantically consistent then metadata can, in some instances, be used to modify the data so that it will be meaningful to the application.

The use of metadata in large statistical databases has been described by McCarthy and others [2,4,5,6,7]. These implementations provide the ability to access and manipulate semantic information not stored in the database. Unlike these authors, we examine the use of data abstraction and encapsulation in metadata representation. Specifically, we describe a representation of metadata using abstract data types (ADTs) and operators. We show how this representation can be used to simplify schema integration in heterogeneous database systems.

We begin by describing the use of metadata in a simplified architecture we call the *source-receiver* model. As shown in Figure 1, the database (source) supplies data used by the application (receiver). The uses for metadata in the interface between a database and application are identical to the uses for metadata in the interface between a global schema representing a set of heterogeneous databases and an application. In Section 2 we describe the use of abstract data types and operators as metadata for the interface between an application and a database. Then in Section 3 we show how metadata across this interface can be used for semantic reconciliation between the application the data source. In Section 4 we describe how metadata is used in the interface between the local databases and the global schema to simplify schema integration. This interface permits some local database autonomy while allowing the global schema to provide an application with a meaningful data set. Finally, we present our conclusions and describe plans for future research.

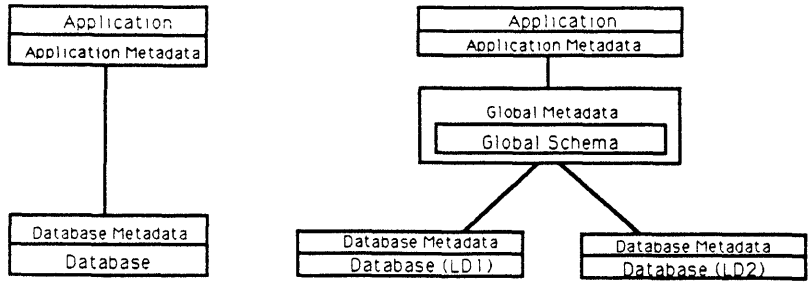

Figure 1: Systems Architecture Using Metadata

2 Abstract Data Types As Metadata

In this section we describe the use of abstract data types to detect and resolve problems associated with changing database semantics in a source-receiver model. Generally, detailed data semantics are provided in printed documents or manuals. These manuals assist the analyst in understanding the data and developing meaningful applications. However, the meaning of the data may change over time; the analyst cannot always be aware of and account for these changes. As the semantics of the data change, the data may no longer be meaningful to the application. For example, an application that uses the unemployment rate to estimate interest rates may have been adversely affected when the government decided to change the unemployment rate calculation to include military personnel.

As an example, consider a data source that provides the trade price for a variety of financial instruments. The schema of the relation containing this data is shown in Figure 2 along with a few sample records. Each record contains the name and type of the instrument being traded, the exchange that the instrument was traded on, and the price of the trade.

Instrument_Type	Instrument_Name	Exchange	Trade_Price
Equity	IBM	NYSE	115.25
Equity	Telecom SP	Madrid SE	1107.25

Figure 2: The FINANCE Relation

A query that requests the price of Telecom SP will return the value 1107.25. Even in this simple relation, the *natural* interpretation of this value might not provide a complete understanding of the data. For example, this relation does not report all trade prices in US currency. Rather, prices are given in the currency of the exchange. The trade price for most equities represents the latest price except for equities traded on the Madrid Stock Exchange where trade price represents the latest nominal price. Because of these semantic complications, the system should provide a means for representation and access to both the trade price value and its associated metadata.

One way to represent this metadata is to extend the traditional database schema definition to include virtual fields. For example, the relation in Figure 2 could be extended to include attributes such as *Units of Currency* and *Trade Price Description.* Values for these fields could be assigned procedurally during data access. However, changing the database

Abstype trade_price *with*

get-status	: trade_price→ trade_price_status*
get-currency	: trade_price→currency*
get-price	: trade_price→real
times	: trade_price#real→trade_price
update-status	: trade_price#trade_price_status* →trade_price
update-curr	: trade_price#currency* →trade_price
update-price	: trade_price#real→trade_price

is

pack <trade_price_status*, currency*, real>
λyϵ<trade_price_status*, currency*, real>.<fst(y)>
λyϵ<trade_price_status*, currency*, real>.<snd(y)>
λyϵ<trade_price_status*, currency*, real>.<thd(y)>
λ<y,z>ϵ<trade_price_status*, currency*, real>#real.<fst(y), snd(y), thd(y) × z>
λ<y,z>ϵ<trade_price_status*, currency*, real>#trade_price_status*.<z, snd(y), thd(y)>
λ<y,z>ϵ<trade_price_status*, currency*, real>#currency*.<fst(y), z, thd(y)>
λ<y,z>ϵ<trade_price_status*, currency*, real>#real.<fst(y), snd(y), z>

Enumeration types

currency*	= {dollar, pasetas, pounds,...}
trade_price_status*	= {latest_price, latest_nominal_price,...}

Figure 3: Abstract Data Type - **trade_price**

schema to represent metadata in this way is awkward and does not simply allow for the use of numerous types of metadata in both schema integration and data processing. Alternatively, we can supplement the conventional schema with abstract data types and a set of operations on those types.

As an example of this use of ADTs consider the semantics of the Trade_Price attribute. For this attribute we can define an abstract data type, **trade_price**, as shown in Figure 3. This abstract type is defined using the SOL [8] syntax, where the list of names bound by the declaration are separate from the definition of the data algebra. The expression beginning with *pack* and running to the end of this example is the definition of the data algebra. This ADT contains a number of operations on the type **trade_price**. Primitives *fst, snd* and *thd* return the values in the first, second and third fields of an instance of the type. For example, the operator *get-currency* returns a value of type **currency*** when applied to a value of type **trade_price**. The *update* operators are used to enter data into the fields of an instance of type **trade_price**.

Next we define operators on a set of abstract data types. These operators assign metadata values to the abstract types. For example, Figure 4 contains the definition of the Trade Price Operator (**t_p_op**). Operator **t_p_op** assigns metadata according to the semantics described for the Trade_Price attribute. Additional abstract types referenced by this operator are shown in Figure 5. For simplicity, update operations on these types are omitted.

```
t_p_op          : instrument_type#exchange#trade_price → trade_price

λ<x,y,z>∈instrument_type#exchange#trade_price
                (get-ins_type x) = Equity
                (get-ex_name y) = Madrid SE
                  .(update-status (update-curr z "pesetas") "latest_nominal_price")

λ<x,y,z>∈instrument_type#exchange#trade_price
                  .(update-status (update-curr z (get-currency y)) "latest_price")
```

Figure 4: Trade Price Operator

Operator **t_p_op** contains two operations. These operations form a hierarchy; the operations with more specific preconditions are placed first in the definition of the operator. Starting at the top of the operator definition, the operation applied is the first one whose preconditions are satisfied.

Operator **t_p_op** takes arguments of type **instrument_type**, **exchange** and **trade_price** and returns a value of type **trade_price**. This operator assigns values of **currency*** and **trade_price_status*** to an instance of type **trade_price**.

3 Using Metadata for Semantic Reconciliation

In this section we demonstrate how ADTs and operators can be used for semantic reconciliation between an application (receiver) and a database (source). Semantic reconciliation is part of the process that ascertains logical connectivity between the database and the application. More precisely, we would like to use this form of metadata to resolve the following questions:

1. Does the database provide data that is semantically meaningful to the application?
2. If the database does not supply semantically meaningful data, then is there a set of operators that can be applied to the existing data to supply data with the required semantics?
3. Is the application affected by a change in the database semantics?

As an example of how metadata can be used to resolve the first question, let the set of ADTs in the database metadata be those described in the previous section. Assume that an application has an identical set of abstract data types, [1] but the operator **t_p_op** is replaced by the operator **app_t_p_op1** as shown in Figure 6. Determining if the database will supply

[1] The set of ADTs define the type of metadata associated with each attribute. We make the simplifying assumption that the ADTs are identical in both the database and the application so that we can concentrate on the use of operators in semantic reconciliation.

Abstype exchange *with*
get-ex_name : exchange→exchange*
get-country : exchange→country*
get-currency : exchange→currency*
is
pack <exchange*, country*, currency*>
λyϵ<exchange*, country*, currency*>. <fst(y)>
λyϵ<exchange*, country*, currency*>. <snd(y)>
λyϵ<exchange*, country*, currency*>. <thd(y)>

Abstype instrument_type *with*
get-ins_type : instrument_type→instrument_type*
is
pack <instrument_type*>
λyϵ<instrument_type*>. <fst(y)>

Enumeration types
exchange* = {NYSE, Madrid SE, London SE,...}
country* = {US, Spain, England,...}
instrument_type* = {Equity, Future, Bond,...}

Figure 5: Additional Abstract Types

meaningful data involves comparing these operators. If, under the preconditions defined for the operations in **app_t_p_op1**, the database operator would make assignments to the trade_price ADT identical to those made by the application operator then we say that the database operator *subsumes* the application operator. If the set of database operators subsume the set of application operators, then the metadata establishes semantic consistency between the application and the database (i.e., the database will supply meaningful data).

In this example, the trade price values supplied by the database will be meaningful if the database operator **t_p_op** *subsumes* the application operator **app_t_p_op1**. If we restrict the domain of operator **t_p_op** to the preconditions found in operator **app_t_p_op1** then we consider only the operation for equities traded on the Madrid Stock Exchange. We can see that under these preconditions the database operator will assign the same metadata values as the application.

There is a different result if the application operator for trade price is that shown in Figure 7 (**app_t_p_op2**). As represented in the operator **t_p_op**, the database supplies trade price in pounds for equities traded on the London Exchange (i.e., the local currency for that exchange). However, the application expects trade price for these equities to be in dollars. The database operator does not subsume this application operator. Unless the semantics of the data can be changed, the application must be alerted that the database cannot supply meaningful data.

The database metadata can contain *conversion* operators. If the set of database operators does not subsume the set of application operators, then it is possible for the database

app_t_p_op1 : instrument_type#exchange#trade_price → trade_price

λ<x,y,z>∈instrument_type#exchange#trade_price
(get-ins_type x) = Equity
(get-ex_name y) = Madrid SE
.(*update-status* (*update-curr* z "pesetas") "latest_nominal_price")

Figure 6: Application Trade Price Operator

app_t_p_op2 : instrument_type#exchange#trade_price → trade_price

λ<x,y,z>∈instrument_type#exchange#trade_price
(get-ins_type x) = Equity
(get-ex_name y) = London SE
.(*update-status* (*update-curr* z "dollars") "latest_nominal_price")

Figure 7: Application Trade Price Operator - Conflict

to use these conversion operators to supply meaningful data. Such an operator is shown in Figure 8. This operator takes a value of type **trade_price** and a value of type **currency*** and returns a value of type **trade_price** with the value of currency* in the second field and the price converted to that currency. The function *exchange-rate* supplies the currency conversion factor.

Though the operator **t_p_op** does not subsume the operator **app_t_p_op2** shown in Figure 7, the operator **convert_currency_op** can be used to convert the currency of the trade price in the database to the currency desired by the application. Together operators **t_p_op** and **convert_currency_op** can be used to provide the application with meaningful data. A *database metadata manager* is needed to identify combinations of operators that can be used to provide the correct data semantics. The application of operators is controlled by the type specifications for the operator. The combination of operators selected for a conversion not only do type conversion but also define routines that can be used to convert the data.

These solutions for determining semantic consistency can also be used in a system where the database (or application) semantics are allowed to change. In such an environment we would like to determine if the application can continue to run in the presence of these changes. For example, the database may change the currency used to report equities traded on the London Exchange from local currency to dollars. For the application metadata shown in Figure 6, this change in the database metadata does not affect the application. Operator subsumption still holds. The application can run unaffected by the change in the database.

```
convert_currency_op  : trade_price#currency* → trade_price

λ<y,z>∈trade_price#currency*
                    .(times < <fst(y),z, thd(y)> exchange-rate(snd(y), z) > )
```

Figure 8: Currency Conversion

Even if changes are made in data semantics, metadata operators can be used to determine if the application is receiving meaningful data.

In this section we described the use of ADTs and operators for use in semantic reconciliation between an application and a database. In some instances, the database metadata manager must identify conversion operators so that the database can adapt to the application's requirements. In the next section we describe how these methods apply to heterogeneous database systems.

4 Schema Integration Using Metadata

The source-receiver model was chosen because of its simplicity. The same model can be used to represent the interaction between an application and a global schema representing an integrated set of local databases. In this configuration, the global schema is responsible for providing meaningful data to the application. Through the use of operators and abstract data types the global schema can determine if semantic changes in the underlying database will affect the application. In this section we describe how metadata descriptions for the local databases are used in schema integration. We propose that operators and ADTs describing the semantics of the local database be included in the local database interface (LDI).

As an example of the use of metadata in the LDI assume the architecture shown in Figure 1. Let the two local databases, LD1 and LD2, be described by the set of abstract data types in Section 2 but let LD1 contain the operator shown in Figure 6 and LD2 contain the operator shown in Figure 7. The integration of these two operators is shown in Figure 9 and represents an integrated view for trade price. Using this integrated view, applications can be designed and additional operators can be included for use in data conversion.

Metadata for the interface between the local database and the global schema can be used to resolve a number of domain mapping problems. For example, LD1 and LD2 contain trade price data in a certain currency. However, if the operator shown in Figure 8 can be included in the global metadata, then a change in the currency reported by the local database would not affect an application. Using the currency conversion operator the global schema is able to present the data in any currency. Similarly, if the value of trade price has an associated scaling factor then it could be included in the definition of the abstract type **trade_price**. Operator **t_p_op** could be rewritten to include the assignment of the scaling factor. Then data with different scaling values can be normalized by accessing the scaling factors for that data. In addition to these types of conversions, metadata can be used to

```
app_t_p_op3     : instrument_type#exchange#trade_price → trade_price

λ<x,y,z>∈instrument_type#exchange#trade_price
                (get-ins_type x) = Equity
                (get-ex_name y) = Madrid SE
                  .(update-status (update-curr z "pesetas") "latest_nominal_price")

λ<x,y,z>∈instrument_type#exchange#trade_price
                (get-ins_type x) = Equity
                (get-ex_name y) = London SE
                  .(update-status (update-curr z "pounds") "latest_nominal_price")
```

Figure 9: Global Integration of Local Database Operators

attach additional information about the data. For example, the operator shown in Figure 9 tags all prices from the Madrid Exchange as the *latest nominal price* and prices from the London Exchange as the *latest price*. Then the global interface may contain rules for the use of values with different tags (e.g., it may not be possible to sum two trade prices when one has a *latest_nominal_price* tag and the other has a *latest_price* tag).

Metadata at the local database interface is used to simplify the integration process and permits limited database autonomy. Metadata at the global interface simplifies semantic reconciliation between the integrated database and the application. Finally, metadata at the global level is useful in supplying a user with information about data semantics.

5 Summary

In this report we present a metadata facility for semantic reconciliation between a data source and a receiver. When semantic connectivity is established, the receiver can be certain that the data supplied has the expected meaning. Results from the use of this model are directly applicable to the interface between an application and a global schema representing the integration of set of local databases. Metadata between the application and the global schema and between the global schema and the local databases provide a means for determining if the application is affected by changes in the underlying databases.

Metadata is used to determine if changes in local database semantics affect the application. If they do not, the application continues execution. If changes in the local databases interfere with the expectations of the application, then the application must be alerted. Conversion operators can be included in the global metadata. These operators allows the global interface to supply meaningful data even when the basic set of operators can not. These conversion operators help to add a level of autonomy for the local databases; the application is further insulated from changes in the local database semantics.

As part of the CIS/TK implementation we are considering using an object-oriented approach to schema integration using metadata. Objects provide the data abstraction and encapsulation present in ADTs and operators, but further investigation is needed to extend

this work to an object-oriented implementation.

Additional work is needed to define the integration of ADTs, operators, and conventional query languages. Query languages and processing techniques must be developed that permit access to data and metadata.

The need to express and manipulate metadata is important in both homogeneous and heterogeneous database systems. Previously, the use of metadata has been examined primarily for retrieval in large statistical databases. This report describes how metadata can be used in the development of intelligent systems that can automatically react to changes in database semantics.

References

[1] P. Dwyer and K. Kasravi. A heterogeneous distributed database management system (ddts/ram). In *Honeywell Report CSC-86-7:8216*, 1986.

[2] A. Goldfine and P. Konig. *A Technical Overview of the Information Resource Dictionary System (Second Edition). NBSIR 88-3700*, National Bureau of Standards, 1988.

[3] T. Landers and R. Rosenberg. An overview of multibase. In *Distributed Data Bases*, pages 153–183, North Holland, 1982.

[4] M. H. Law. *Guide to Information Resource Dictionary System Applications: General Concepts and Strategic Systems Planning. 500-152*, National Bureau of Standards, 1988.

[5] J. McCarthy. Information systems design for material properties data. In *Proceedings of the First International Symposium on Computerization and Networking of Material Property Databases*, American Society for Testing and Materials, Philadelphia, 1987.

[6] J. McCarthy. Metadata management for large statistical database. In *Proceedings of the Eight International Conference on Very Large Database Systems*, pages 470–502, Mexico City, 1988.

[7] J. McCarthy. Scientific information = data + meta-data. In *Database Management: Proceedings of the Workshop November 1-2, U.S. Navy Postgraduate School, Monterey, California*, Department of Statistics Technical Report, Stanford University, 1988.

[8] J.C. Mitchell and G.D. Plotkin. Abstract types have existential types. *ACM Trans. on Programming Languages and Systems*, 10(3):470–502, 1988. Preliminary version appeared in *Proc. 12-th ACM Symp. on Principles of Programming Languages*, 1985.

[9] S. Madnick and R. Wang et al. CISL : Composite answers from disparate information systems. In *Submitted to the IEEE Workshop on Heterogeneous Database Systems*, 1989.

[10] M. et al Templeton. Mermaid - a front-end to distributed heterogeneous databases. In *Proceedings of the IEEE*, pages 695–708, 1987.

[11] R. Wang and S. Madnick, editors. *Connectivity Among Information Systems.* Volume 1 of *Composite Information Systems (CIS) Project*, Massachusetts Institute of Technology, 1988.